Univerza v Ljubljani

Prvenstvo UNIVERZE V LJUBLJANI v programiranju

2002

1. krog

Pokrovitelji:

UNIVERZA V LJUBLJANI
Fakulteta za matematiko in fiziko

Prvenstvo v programiranju 2002

1. krog 27. maj 2002

Prvenstvo v programiranju 2002

1. krog 27. maj 2002

NAVODILA

Prijava: Prijavite se lahko le na en računalnik. Če bodo med tekmovanjem z njim slučajno težave, javite vodstvu tekmovanja. V primeru, da boste hkrati prijavljeni na dveh računalnikih, bo ekipa diskvalificirana.

Uporabniško ime je sestavljeno iz besede acm in številke računalnika.

Primer: uporabniško ime na računalniku št. 1 je: acm01

uporabniško ime na računalniku št. 12 je: acm12

Geslo je sporočeno pred začetkom tekmovanja.

Vaše domače področje na disku je `/home/acmXX` kjer je XX številka računalnika

Priprava delovnega okolja

Po uspešni prijavi morate odpreti dva terminala in program Netscape.

Terminal št.1: v njem boste pisali programe, prevajali programe in pošiljali programe sodniku.

Terminal št.2: v njem boste dobivali obvestila o pravilnosti rešitev in pošiljali morebitna vprašanja sodniku.

Program Netscape: bo omogočal pregled trenutne uspešnosti ekip

Pisanje programov:

Na voljo so urejevalniki besedil: *vi, joe, jed, mcedit, emacs*, *gedit in kedit*

Ime programov je točno določeno pri opisu nalog. Pazite na `return 0,` ki mora zaključiti vsak uspešno izveden C program.

Prevajanje programov:

Za prevajanje programov morate uporabljati naslednje prevajalnike:

```
C       (*.c)        gcc    primer:   gcc -o program program.c
C++     (*.cpp)      g++    primer:   g++ -o program program.cpp
Pascal (*.pas)       gpc    primer:   gpc -o program program.pas
```

Pošiljanje programov:

Za pošiljanje programov sodniku uporabljate ukaz `submit`.

Primer: `submit program.c`

Sporočila o napakah – PE in WA

Zaradi različnih razlogov je zelo težko razlikovati med `Presentation Error` in `Wrong Answer`. Ker preverjanje poteka več ali manj avtomatsko, program pogosto ne zna razlikovati med tem, ali ste le pozabili piko v odgovoru, ali pa ste napačno zračunali. Zato tudi `Wrong Answer` včasih lahko pomeni

le manjkajoč znak v odgovoru. Praviloma naj bi sporočilo `PE` dobili le v primeru napačno postavljenih presledkov ali ločil in praznih vrstic, ni pa to nujno!

Komunikacija sodniki – ekipa:

V terminalu 2 izvedete ukaz: `telnet svarun`

Za prijavo na računalnik svarun uporabite isto uporabniško ime in geslo, kot ste ga uporabili za prijavo na vaš računalnik. V tem terminalskem oknu se bodo izpisovala sodnikova sporočila vam. Če želite poslati sporočilo sodniku, morate v tem terminalu napisati ukaz:

```
write sodnik
```

in potem sporočilo, ki ga zaključite s kombinacijo tipk CTRL + D na začetku prazne vrstice. Prosimo, da **ne** pošiljate sporočil tipa `... Sedaj smo poslali program ...` in `... Naš program zagotovo dela, pa dobimo WA ...` Ekipa, ki bo pošiljala tovrstna sporočila, je lahko tudi diskvalificirana.

Trenutni rezultati:

Program Netscape: bo omogočal pregled trenutne uspešnosti ekip

Po zagonu programa greste na naslov:

http://svarun.fmf.uni-lj.si/index.html

Kadar želite videti trenutno stanje, morate narediti »reload«.

Pritožbe:

So možne le takoj po zaključku tekmovanja. Kasneje se rezultati lahko spremenijo le v primeru drastične sodniške napake.

Preklapljanje med angleško in slovensko tipkovnico:

(Velja samo za okolje X windows)

Prednastavljena je slovenska tipkovnica. Za preklop na angleško tipkovnico v terminalu izvedete ukaz

```
setxkbmap us
```

za preklop nazaj na slovensko pa:

```
setxkbmap sl
```

Izbrana tipkovnica potem velja za celotno X okolje.

Pinball Simulation

Program PINBALL.C, PINBALL.CPP, PINBALL.PAS

Background

Simulation is an important application area in computer science involving the development of computer models to provide insight into real-world events. There are many kinds of simulation including (and certainly not limited to) discrete event simulation and clock-driven simulation. Simulation often involves approximating observed behavior in order to develop a practical approach.

This problem involves the simulation of a simplistic *pinball* machine. In a pinball machine, a steel ball rolls around a surface, hitting various objects (*bumpers*) and accruing points until the ball ``disappears'' from the surface.

The Problem

You are to write a program that simulates an idealized pinball machine. This machine has a flat surface that has some obstacles (bumpers and walls). The surface is modeled as an m×n grid with the origin in the lower-left corner. Each bumper occupies a grid point. The grid positions on the edge of the surface are walls. Balls are shot (appear) one at a time on the grid, with an initial position, direction, and lifetime. In this simulation, all positions are integral, and the ball's direction is one of: up, down, left, or right. The ball bounces around the grid, hitting bumpers (which accumulates points) and walls (which does not add any points). The number of points accumulated by hitting a given bumper is the *value* of that bumper. The speed of all balls is one grid space per timestep. A ball ``hits'' an obstacle during a timestep when it would otherwise move on top of the bumper or wall grid point. A hit causes the ball to ``rebound'' by turning right (clockwise) 90 degrees, without ever moving on top of the obstacle and without changing position (only the direction changes as a result of a rebound). Note that by this definition sliding along a wall does not constitute ``hitting'' that wall.

A ball's lifetime indicates how many time units the ball will live before disappearing from the surface. The ball uses one unit of lifetime for each grid step it moves. It also uses some units of lifetime for each bumper or wall that it hits. The lifetime used by a hit is the *cost* of that bumper or wall. As long as the ball has a positive lifetime when it hits a bumper, it obtains the full score for that bumper. Note that a ball with lifetime one will ``die'' during its next move and thus cannot obtain points for hitting a bumper during this last move. Once the lifetime is non-positive (less than or equal to zero), the ball disappears and the game continues with the next ball.

Input

Your program should simulate one game of pinball. There are several input lines that describe the game. The first line gives integers m and n, separated by a space. This describes a cartesian grid where $1 \le x \le m$ and $1 \le y \le n$ on which the game is ``played''. It will be the case that $2 < m < 51$ and $2 < n < 51$. The next line gives the integer cost for hitting a wall. The next line gives the number of bumpers, an integer $p \ge 0$. The next p lines give the x position, y position, value, and cost, of each bumper, as four integers per line separated by space(s). The x and y positions of all bumpers will be in the range of the grid. The value and cost may be any integer (i.e., they may be negative; a negative cost *adds* lifetime to a ball that hits the bumper). The remaining lines of the file represent the balls. Each line represents one ball, and contains four integers separated by space(s): the initial x and y position of the ball, the direction of movement, and its lifetime. The position will be in range (and not on top of any bumper or wall). The

direction will be one of four values: 0 for increasing x (right), 1 for increasing y (up), 2 for decreasing x (left), and 3 for decreasing y (down). The lifetime will be some positive integer.

Output

There should be one line of output for each ball giving an integer number of points accumulated by that ball in the same order as the balls appear in the input. After all of these lines, the total points for all balls should be printed.

Sample Input

```
4 4
0
2
2 2 1 0
3 3 1 0
2 3 1 1
2 3 1 2
2 3 1 3
2 3 1 4
2 3 1 5
```

Sample Output

```
0
0
1
2
2
5
```

Primary Arithmetic

Program PRIMARY.C, PRIMARY.CPP, PRIMARY.PAS

Children are taught to add multi-digit numbers from right-to-left one digit at a time. Many find the "carry" operation - in which a 1 is carried from one digit position to be added to the next - to be a significant challenge. Your job is to count the number of carry operations for each of a set of addition problems so that educators may assess their difficulty.

Input

Each line of input contains two unsigned integers less than 10 digits. The last line of input contains 0 0.

Output

For each line of input except the last you should compute and print the number of carry operations that would result from adding the two numbers, in the format shown below.

Sample Input

```
123 456
555 555
123 594
0 0
```

Sample Output

```
No carry operation.
3 carry operations.
1 carry operation.
```

Climbing Trees

Program TREES.C, TREES.CPP, TREES.PAS

Given a sequence of *child-parent* pairs, where a pair consists of the child's name followed by the (single) parent's name, and a list of query pairs also expressed as two names, you are to write a program to determine whether the query pairs are related. If the names comprising a query pair are related the program should determine what the relationship is. Consider academic advisees and advisors as exemplars of such a single parent genealogy (we assume a single advisor, i.e., no co-advisors).

In this problem the child-parent pair p q denotes that p is the child of q. In determining relationships between names we use the following definitions:

- p is a *0-descendent* of q (respectively *0-ancestor*) if and only if the child-parent pair p q (respectively q p) appears in the input sequence of child-parent pairs.
- p is a *k-descendent* of q (respectively *k-ancestor*) if and only if the child-parent pair p r (respectively q r) appears in the input sequence and r is a (k−1)-descendent of q (respectively p is a (k−1)-ancestor of r).

For the purposes of this problem the relationship between a person p and a person q is expressed as exactly one of the following four relations:

1. child - grand child, great grand child, great great grand child, *etc.*

   By definition p is the ``child" of q if and only if the pair p q appears in the input sequence of child-parent pairs (i.e., p is a 0-descendent of q); p is the ``grand child" of q if and only if p is a 1-descendent of q; and

   p is the ``great great ... great grand child'' of q *(``great'' repeated n times)*

   if and only if p is an (n+1)-descendent of q.

2. parent - grand parent, great grand parent, great great grand parent, *etc.*

   By definition p is the ``parent" of q if and only if the pair q p appears in the input sequence of child-parent pairs (i.e., p is a 0-ancestor of q); p is the ``grand parent" of q if and only if p is a 1-ancestor of q; and

   p is the ``great great ... great grand parent'' of q *(``great'' repeated n times)*

   if and only if p is an (n+1)-ancestor of q.

3. cousin - 0^{th} cousin, 1^{st} cousin, 2^{nd} cousin, *etc.*; cousins may be once removed, twice removed, three times removed, *etc.*

   By definition p and q are ``cousins" if and only if they are related (i.e., there is a path from p to q in the implicit undirected parent-child tree). Let r represent the least common ancestor of p and q (i.e., no descendent of r is an ancestor of both p and q), where p is an m-descendent of r and q is an n-descendent of r.

Then, by definition, cousins p and q are ``$k^{th}$ cousins" if and only if k = min(n, m), and, also by definition, p and q are ``cousins removed j times" if and only if j = | n – m | .

4. sibling - 0^{th} cousins removed 0 times are ``siblings" (they have the same parent).

The Input

The input consists of parent-child pairs of names, one pair per line. Each name in a pair consists of lower-case alphabetic characters or periods (used to separate first and last names, for example). Child names are separated from parent names by one or more spaces. Parent-child pairs are terminated by a pair whose first component is the string ``*no.child*". Such a pair is NOT to be considered as a parent-child pair, but only as a delimiter to separate the parent-child pairs from the query pairs. There will be no circular relationships, i.e., no name p can be *both* an ancestor and a descendent of the same name q.

The parent-child pairs are followed by a sequence of query pairs in the same format as the parent-child pairs, i.e., each name in a query pair is a sequence of lower-case alphabetic characters and periods, and names are separated by one or more spaces. Query pairs are terminated by end-of-file.

There will be a maximum of 300 different names overall (parent-child and query pairs). All names will be fewer than 31 characters in length. There will be no more than 100 query pairs.

The Output

For each query-pair p q of names the output should indicate the relationship p *is-the-relative-of* q by the appropriate string of the form

- child, grand child, great grand child, great great ... great grand child
- parent, grand parent, great grand parent, great great ... great grand parent
- sibling
- n cousin removed m
- no relation

If an m-cousin is removed 0 times then only *m cousin* should be printed, i.e., *removed 0* should NOT be printed. Do not print *st, nd, rd, th* after the numbers.

Sample Input

```
alonzo.church oswald.veblen
stephen.kleene alonzo.church
dana.scott alonzo.church
martin.davis alonzo.church
pat.fischer hartley.rogers
mike.paterson david.park
dennis.ritchie pat.fischer
hartley.rogers alonzo.church
les.valiant mike.paterson
bob.constable stephen.kleene
david.park hartley.rogers
no.child no.parent
stephen.kleene bob.constable
hartley.rogers stephen.kleene
les.valiant alonzo.church
les.valiant dennis.ritchie
dennis.ritchie les.valiant
pat.fischer michael.rabin
```

Sample Output

```
parent
sibling
great great grand child
1 cousin removed 1
1 cousin removed 1
no relation
```

Following Orders

Program ORDERS.C, ORDERS.CPP, ORDERS.PAS

Order is an important concept in mathematics and in computer science. For example, Zorn's Lemma states: ``a partially ordered set in which every chain has an upper bound contains a maximal element.'' Order is also important in reasoning about the fix-point semantics of programs. This problem involves neither Zorn's Lemma nor fix-point semantics, but does involve order.

Given a list of variable constraints of the form `x < y`, you are to write a program that prints all orderings of the variables that are consistent with the constraints.

For example, given the constraints `x < y` and `x < z` there are two orderings of the variables x, y, and z that are consistent with these constraints: `x y z` and `x z y`.

The Input

The input consists of a sequence of constraint specifications. A specification consists of two lines: a list of variables on one line followed by a list of contraints on the next line. A constraint is given by a pair of variables, where `x y` indicates that `x < y`.

All variables are single character, lower-case letters. There will be at least two variables, and no more than 20 variables in a specification. There will be at least one constraint, and no more than 50 constraints in a specification. There will be at least one, and no more than 300 orderings consistent with the contraints in a specification.

Input is terminated by end-of-file.

The Output

For each constraint specification, all orderings consistent with the constraints should be printed. Orderings are printed in lexicographical (alphabetical) order, one per line. Characters on a line are separated by whitespace.

Output for different constraint specifications is separated by a blank line.

Sample Input

```
a b f g
a b b f
v w x y z
v y x v z v w v
```

Sample Output

```
abfg
abgf
agbf
gabf

wxzvy
wzxvy
xwzvy
xzwvy
zwxvy
zxwvy
```

Shoes maker's problem

Program SHOES.C, SHOES.CPP, SHOES.PAS

Shoemaker has N jobs (orders from customers) which he must make. Shoemaker can work on only one job in each day. For each i_{th} job, it is known the integer T_i ($1<=T_i<=1000$), the time in days it takes the shoemaker to finish the job. For each day of delay before starting to work for the i_{th} job, shoemaker must pay a fine of S_i ($1<=S_i<=10000$) cents. Your task is to help the shoemaker, writing a programm to find the sequence of jobs with minimal total fine.

The Input

First line of input for each case contains an integer N ($1<=N<=1000$). The next N lines each contain two integers: the time and fine of each task in order. Time is never less than 1. A value N = 0 indicates end of input.

The Output

For each case, your program should print the case number, followed by the sequence of jobs with minimal fine. Each job should be represented by its number in input. All integers should be placed on only one output line and separated by one space. If multiple solutions are possible, print the first lexicographically.

Sample Input

```
3
1 5
2 4
3 3
4
3 4
1 1000
2 2
5 5
0
```

Sample Output

```
Case #1: 1 2 3
Case #2: 2 1 3 4
```